

# Execute command



Execute command is a default Switch app that executes a system command or a console application with the specified command-line arguments and output mechanisms.

## Keywords

Keywords can be used with the search function above the Elements pane.

The keywords for the **Execute command** element are:

- command
- console
- third-party
- CLI



## Connections


Execute command supports outgoing traffic light connections of the following types:

- Success out: carries successfully passed jobs.
- Error out (optional): carries jobs that failed during command execution.

## Properties

Properties	Description
Element type	The flow element type: Execute command. This property is useful to identify renamed flow elements. It cannot be changed.
Name	The name of the flow element displayed in the canvas
Description	A description of the flow element displayed in the canvas. This description is also shown in the tooltip that appears when moving your cursor over the flow element
Execution mode	This property helps you to configure the tool to handle jobs in parallel (concurrent) or serial mode: <ul style="list-style-type: none"><li>• <b>Concurrent:</b> Choose this option to enable the tool to process jobs simultaneously through multiple instances of the <b>Execute Command</b> tool while using the same application.</li></ul>

Properties	Description
	 <b>Note:</b> <ul style="list-style-type: none"> <li>The application you choose should support concurrent processing.</li> <li>Use the <b>Concurrent processing tasks</b> property in the <b>User Preferences &gt; Processing</b> tab to set the number of concurrent processes.</li> </ul> <hr/> <ul style="list-style-type: none"> <li><b>Serialized:</b> Choose this option to enable the tool to process jobs using only a single instance of the <b>Execute Command</b> tool.</li> </ul>
Command or path	<p>The system command to be executed (only allowed for commands from the operating system!), or the <b>absolute path</b> to the console application to be invoked.</p> <hr/>  <b>Note:</b> If you don't know the location of the application, search for it in the documentation of the application or search for the application on your disk. On Windows, applications are usually installed under C:\Program Files\ or C:\Program Files (x86)\. On Mac, you can use Spotlight to search in the system files (see <a href="http://support.apple.com/kb/HT4355">http://support.apple.com/kb/HT4355</a> ) .
Arguments	<p>The remainder of the command line; the following substitutions are made in the specified text:</p> <ul style="list-style-type: none"> <li>%1: the absolute path of the input file/job folder</li> <li>%2: the absolute path of the output file/job folder (which does not yet exist) including filename and extension</li> <li>%3: the absolute path of a folder (which does exist) in which output or temp files may be placed</li> </ul> <p>All of these substitutions work even if "Output" is set to None or Original (in which case any generated output is discarded)</p> <p><b>Placeholders (%1, %2, %3) and argument values that may contain spaces (such as file paths) must be enclosed in quotes!</b></p>
Output	<p>Determines how the command generates its output (the tool automatically moves the output to the outgoing connection):</p> <ul style="list-style-type: none"> <li>None: the command generates no output (or any generated output is discarded) and nothing is moved to the outgoing connection</li> <li>Original: the command generates no output (or any generated output is discarded) and the incoming job is moved to the outgoing connection</li> </ul>

Properties	Description
	<ul style="list-style-type: none"> <li>Updated input job: the command updates or overwrites the incoming job in place</li> <li>Result next to input job: the command places the output next to the input job (i.e. in the same folder)</li> <li>File at path: the command places a single output file at the absolute path specified by %2</li> <li>Folder at path: the command places a single output job folder at the absolute path specified by %2</li> <li>Result in folder: the command places the output inside the existing folder at absolute path %3</li> </ul> <div>  <b>Note:</b> When you enter %3 in the Arguments property, the tool creates a temporary folder named ExecuteCommandTemp. Switch sends the entire ExecuteCommandTemp folder (and not just the contents of the folder) as job folder to the outgoing connection. </div>
<i>Copy input job</i>	<p>If set to yes, the incoming job is copied to a temporary location before the command is triggered; this must be turned on in cases where the command modifies the input job or places files in the folder where the input job resides</p> <p>This property is not present for some choices of output because in those cases the incoming job must always be copied</p>
<i>Output job filter</i>	Determines which of the generated files or folders is the output file; Automatic works unambiguously if a single file is generated; otherwise a simple heuristic is used to make an "educated guess"
<i>Output extension</i>	The filename extension of the output job to be provided to the command line (i.e. the substitution of %2); Automatic means use the same extension as the input job
Exit code	<p>Defines how to handle the exit code:</p> <ul style="list-style-type: none"> <li>Set as private data: The exit code is saved as private data in the job with the key specified in the 'Private data key' property.</li> <li>Discard: The exit code is discarded.</li> </ul>
<i>Private data key</i>	<p><i>Only available if Exit code is set to Set as private data</i></p> <p>Custom name for the private data key that stores the exit code of the command.</p>
Stdout	<p>Defines how to handle the standard output stream:</p> <ul style="list-style-type: none"> <li>Attach as dataset: The output stream is saved to a text file and the file is attached to the output job as an opaque dataset with the name specified in the 'Dataset name' property.</li> </ul>

Properties	Description
	<ul style="list-style-type: none"> <li>Discard: The output stream is discarded.</li> </ul>
<i>Dataset name</i>	<p><i>Only available if Stdout is set to Attach as dataset</i></p> <p>Custom name for the dataset that stores the output stream of the command.</p>
Stderr	<p>Defines how to handle the standard error stream:</p> <ul style="list-style-type: none"> <li>Attach as dataset: The error stream is saved to a text file and the file is attached to the output job as an opaque dataset with the name specified in the 'Dataset name' property.</li> <li>Discard: The error stream is discarded.</li> </ul>
<i>Dataset name</i>	<p><i>Only available if Stderr is set to Attach as dataset</i></p> <p>Custom name for the dataset that stores the error stream of the command.</p>
Fail if exit code is	<p>Fails the job (that is, moves it to the problem jobs folder) if the command's exit code is:</p> <ul style="list-style-type: none"> <li>Nonzero</li> <li>Negative</li> <li>In range</li> <li>Outside range</li> <li>Disregard exit code</li> </ul>
<i>Range</i>	<p>A range specified as "n1..n2;n3..n4" where the n's represent a positive or negative integer or are missing to indicate an open end; for example: "-7..-4; 0.."</p>
Fail if stdout contains	<p>Fails the job (that is, moves it to the problem jobs folder) if the regular console output contains the search string or pattern; empty means disregard the regular console output</p>
Fail if stderr contains	<p>Fails the job (that is, moves it to the problem jobs folder) if the console error output contains the search string or pattern; empty means disregard the console error output</p>

## Best practice

We recommend creating a working command in Command Prompt on Windows or Terminal on Mac first, before actually configuring the Execute command flow element.

Keep in mind that **Switch will execute the command directly in the operating system** while Command Prompt and Terminal use a shell. This results in some differences in behavior, the most important one being that you always have to provide the full path to the command line application in Switch, even when this is not necessary in Command Prompt or Terminal!

For complex issues, it may be handy to **enable the logging of debug messages** (see *Switch Preferences: Logging* in the [Switch Reference Guide](#)). The table below gives an overview of the debug messages that will be logged when running the Execute Command Tool:

Debug message	Meaning
Executing	Complete command which Switch will execute after it replaced the placeholders with the actual paths.
Outcode	The outcode/exitcode of the command. Switch decides whether or not the command was successful.
Stdout	Output of the command in case executing the command went well. This is the same output as you would see when executing the command in Command Prompt or Terminal.
Stderr	Output of the command in case of problems. This is the same output as you would see when executing the command in Command Prompt or Terminal.

If none of the above debug messages are generated, the operating system failed to execute the command itself. Check if the path to the application is correct and check if you have the right permissions.

## Redirecting

Redirecting output from a command to a text file is a feature of the shells used in Terminal and Command Prompt and is therefore not possible using the Execute command tool in Switch.

As a workaround, you can create a batch file that uses a shell that supports redirecting. Use this batch file as a command and send input and output paths as arguments to this batch file.

### Example:

*Setup of the Execute Command Element:*

- Command or path: /path/to/bash/script.sh
- Arguments: "%1" "%2"
- Output: File at path

*Contents of the script.sh batch file:*

```
#!/bin/bash
sed "s/\&/\&amp;/g" "${1}" >> "${2}" (This example will replace "&" with "&amp;" in text files)
```

## Best practices

- Sometimes it is useful and easier to **write arguments in the Script Expression dialog**. Below is an example:

```
Command or path: cp
Arguments: Script expression defined:
    "var destPath = "/Users/user/Desktop/dest";
    var path = job.getPath();
    var command = "\"" + path + "\" \"\" + destPath + "\"";
    command;"
```

- Make sure that all the passed arguments are the **real arguments of the specified process**. In the example below, the TASKKILL command is not the command of Application.

```
Command or path: C:/Program Files/Application /Application 1.0/Application/
Application.exe
Arguments:      /t "%1" "option with space" "option with space" TASKKILL /IM
               "Application.exe" <-- WRONG -->
```

A similar mistake would be to write the following in cmd:

```
C:\Users\user>dir "c:\folder1\folder2" taskkill /im Application.exe <-- WRONG -->
```

On the contrary, this is the right way to write it:

```
C:\Users\user>dir "c:\folder1\folder2"          <-- CORRECT -->
C:\Users\user>taskkill /im Application.exe      <-- CORRECT -->
```

- Make sure to use **variables** (%1, %2, %3) correctly. Below is an example:

```
Command or path: tar
Arguments: -zxvf "%1" -C "%3"
```

- Remember to use **quotes** when referring to paths.

```
dir c:/folder/inner folder <-- WRONG -->
dir "c:/folder/inner folder" <-- CORRECT -->
```

- Preferably, use a **bat file/shell script** instead of an executable or an app.

This means that the "Command or path" property is used to select the bat file or shell script and the variable parameters. All other arguments are put in the bat file or shell script. This approach has two advantages:

- It keeps the Arguments property cleaner and easier to read.
- The executable or app in the Execute command is running in a different environment than when used in a command prompt/terminal window. This may be necessary to be able to work with certain environment variables.

Example:

```
Content of the shell script:
#!/bin/bash
export PATH="/cygdrive/c/Program Files/MySQL/MySQL Server 5.7/bin"
echo $PATH
#do everything that you want
#start your application for example
mysql $1

Command or path: <path to bash>
Arguments: <path to sh> <mysql arguments>
```

- You can use **Switch variables in the 'Arguments' property**, because all the variables are evaluated and actual values are passed inside the script; actually, the arguments from the 'Arguments' property become script arguments, accessible inside the script via \$1, \$2 etc. or %1, %2 etc. However, keep in mind that it is useless to write Switch variables inside .sh or .bat scripts, as Switch doesn't work with the script content.
- Use the **Run tool to recheck the command and arguments on Windows**. For example, proceed as follows:

1. Find Run on your system.
2. Put "cmd /K dir <path to the directory>" ( without quotes ) in Run.
3. Press Enter.
4. Recheck that the output is correct.